Contents

[Introduction 2](#_Toc101514875)

[Prerequisites 2](#_Toc101514876)

[Create the RPC project 2](#_Toc101514877)

[Add the NidRPC NuGet to the project 2](#_Toc101514878)

[Device RPC 3](#_Toc101514879)

[Finding RPC calls 3](#_Toc101514880)

[Making an RPC call 4](#_Toc101514881)

[Create and install the the application 4](#_Toc101514882)

# Introduction

This document describes how to create the FR22 RPC application in C#. The source code for the sample can be found at [https://github.com/NordicID/fr22\_samples/tree/master/2. RPC demo](https://github.com/NordicID/fr22_samples/tree/master/2.%20RPC%20demo).

The RPC demo sample application creates an RPC instance and uses it to get the current firmware version and prints it to the application log.

## Prerequisites

* FR22 device with OS version 0.8.0 or later
* fr\_appsigntool files from fr22\_appsigntool directory in <https://github.com/NordicID/fr22_samples/> (requires .NET Core 3.1)
* Visual Studio 2019/2022 with C# support installed
  + It’s possible to use VS Code or the dotnet tool, but this guide assumes Visual Studio is used.
* Knowledge of how the Hello World sample described in the 1. FR22 Hello World.docx works

# Create the RPC project

Create a new RPC demo project in a similar manner as the Hello World sample was created.

# Add the NidRPC NuGet to the project

NidRpc is a C# library available as a NuGet package. NidRpc implements the MQTT based request/response RPC protocol used internally on the FR22. It is available on nuget.org at <https://www.nuget.org/packages/NidRpc/>. Install it to the project by:

* Opening the NuGet package manager (in the Solution explorer, right click on either the project name or *References* and select *Manage NuGet Packages*)
* Select the browse tab and search for nidrpc (making sure nuget.org is the selected package source)

A screenshot of a computer

Description automatically generated with medium confidence

* Select the NidRPC entry and press *Install*, and install NidRpc and its’ dependencies

# Device RPC

## Finding RPC calls

The RPC protocol on the FR22 makes function calls to other processes over MQTT, by mapping a functional call to an MQTT topic. The available plugins on the device can be viewed in the WebUI by navigating to  
*System/Api Docs*, e.g.:

A computer screen capture

Description automatically generated with medium confidence

Plugins are identified by *type/name* tuples, where *type* can be one of *builtin, plugin* and *application*. Plugins of the *builtin* type are made by Nordic ID and are statically include in the firmware (i.e., can only be upgraded by upgrading the firmware). Plugins of the *plugin* type are system plugins made by Nordic ID, a version may be included in the firmware, or it may be installed at runtime (by navigating to *Software/System Plugins*). Plugins of the *application* type can be created by anyone and installed at runtime.

To view the topics available for a plugin, click on the name, for example *builtin/sysinfo*:

A screenshot of a computer

Description automatically generated

As previously mentioned, a topic represents a function call. Arguments are (usually) given as JSON, and the function (usually) returns JSON. To view the documentation for the function, press the topic name, e.g., pressing *api/builtin/sysinfo/versions* above gets:

Graphical user interface, text, application

Description automatically generated

From this we can see that if *api/builtin/sysinfo/versions* is called (with an empty payload), we get back a JSON collection containing name/value pairs with information about the system.

## Making an RPC call

To use the NidRPC library, import the NidRPC namespace defined by it by adding a line with:

using NidRpc;

to the top of Program.cs. Then in Main, create a NidRPC client:

Client rpc = new Client("RPCDemo");

Since NidRPC uses MQTT, every RPC instance needs to be given a unique name, in this case *RPCDemo*. To connect the client, call ConnectAsync on it and await the result.

await rpc.ConnectAsync();

The NidRPC is an asynchronous library and operations need to be awaited; for this reason, the Main signature in the sample has also been changed to async Task.

Finally, to make a call, the Call function is called with the desired topic (in this case *api/builtin/sysinfo/versions*) and the resulting JSON JObject is awaited:

JObject rsp = await rpc.Call("api/builtin/sysinfo/versions");

If there was an error, the JSON JObject response will have an *error* key with a string value describing the error condition, otherwise the JSON for this topic will contain a version key. The rest of the RPC demo sample just prints the version field to the log.

# Create and install the the application

The application is built, created and installed in the same way as the Hello World sample described in the 1. FR22 Hello World.docx document. Note that there are two minor differences:

* This build result of this sample also contains several .dll files (NidRpc.dll and its’ dependencies), these also should be copied from bin\Release to ZipContents\bin
* The sample is automatically started after installation by adding

"default\_autostart": true,

to the “startup” section in meta\manifest.json. Some applications need to be configured before they can be used; in this case it makes sense to leave default\_autostart to the default value (false). In this case the application can be started immediately after installation.

After the application has been installed, the log print can be viewed by again pressing the Log button on the RPCdemo row on the *Software/Applications* page.

Timeline

Description automatically generated with low confidence